a) Expressão de raciocínio algorítmico.

b) Execução automática de um algoritmo por um computador.

1.3 PROGRAMAÇÃO ESTRUTURADA

Basicamente, a Programação Estruturada consiste numa *metodologia de projeto de programas* visando:

- facilitar a *escrita* dos programas;
- facilitar a *leitura* (o entendimento) dos programas;
- permitir a *verificação a priori* dos programas;
- facilitar a *manutenção* e *modificação* dos programas.

O maior problema em grandes sistemas de *software* reside na enorme *complexidade* desses sistemas, cuja apreensão vai geralmente muito além da capacidade intelectual de um ser humano. Entenda-se aqui por complexidade de um sistema uma medida do número de seus componentes e do grau de interação entre eles. Para Dijkstra, o indiscutível iniciador da programação estruturada, ''*a arte de programar consiste na arte de organizar e dominar a complexidade*''.

A idéia básica da Programação Estruturada, que vai ao encontro da mencionada tarefa do programador, é *reduzir a complexidade*, em três níveis:

a) desenvolvimento do programa em diferentes fases por refinamento sucessivo (*desenvolvimento top-down*);

b) decomposição do programa total em *módulos funcionais*, organizados de preferência num *sistema hierárquico*;

c) usando dentro de cada módulo só um número muito limitado de *estruturas básicas de fluxo de controle.*

1.3.1 Desenvolvimento Top-Down

Antes de poder escrever um programa, de fato sempre é necessário um *processo de raciocínio*, que leva de uma análise do problema dado, passando por um algoritmo em termos gerais até um algoritmo detalhado, que consiste em uma seqüência de passos simples que podem ser diretamente expressados em termos de comandos numa linguagem de programação.

Na Programação Estruturada as diferentes fases deste processo de concepção de um programa são fixadas explicitamente por escrito, e não só — como é uso comum — o produto final: o programa na forma em que será submetido à máquina. Obtemos no final não somente *um* programa (na linguagem que foi escolhida para a codificação), mas também uma documentação de todo o processo de solução do problema.

Para as *estruturas de dados* usamos igualmente representações abstratas de acordo com as necessidades do problema, que vão sendo refinadas até chegar à representação implementável na linguagem de programação. Isso permite desenvolver passo a passo o algoritmo-solução e as estruturas de dados em termos das *categorias relevantes ao problema*, e não das peculiaridades de uma linguagem específica de programação.

Cada nova fase desse desenvolvimento ''de cima para baixo'' é obtida por ''*refinamento*'' da fase anterior, até chegar a um nível de detalhamento que permita implementar o algoritmo diretamente na linguagem de programação.

1.3.2 Modularização

Durante a fase de projeto descrita acima, a solução do problema total vai sendo fatorada em soluções de subproblemas, o que permite geralmente dividir o programa em forma natural em módulos com subfunções claramente delimitadas, que podem ser implementados separadamente por diversos programadores de uma equipe.

1.3.3 Estruturas de Controle

A legibilidade e compreensão de cada módulo é enormemente incrementada, proibindo o uso irrestrito de comandos de desvio (GOTO). Estes comandos alteram a seqüência natural de execução do programa desviando para um endereço especificado pelo programa. Deve-se usar, tanto quanto possível, somente algumas estruturas básicas de fluxo de controle: a *seqüência simples*, o *comando condicional* e o *comando repetitivo*, que correspondem a formas de raciocínio intuitivamente óbvias.

1.3.4 Motivação: Problemas na Produção de Software

Antes de passar a discutir as diferentes técnicas da Programação Estruturada, apresentamos alguns dos problemas no desenvolvimento de (grandes) sistemas de programação que levaram ao fim dos anos 60 à chamada *crise do software*, nos países desenvolvidos.

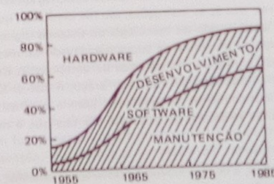


Fig. 1.1 Tendência dos custos de *software/hardware*.

No começo do uso de computadores o fator dominante nos custos da computação era o equipamento eletrônico: o *hardware*. Hoje, 20 anos depois, a situação se inverteu drasticamente. O gráfico acima mostra a tendência da relação dos custos de *software/hardware* de